PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2001-034619
(43)Date of publication of application : 09.02.2001

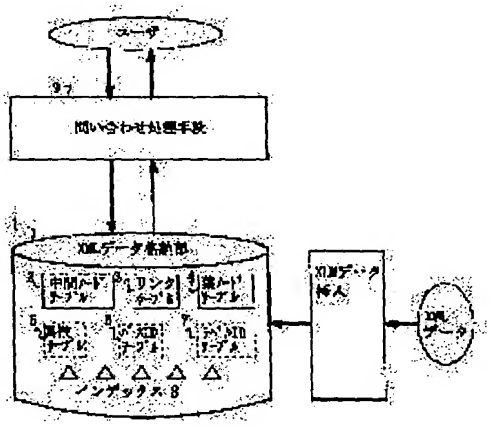
(51)Int.Cl. G06F 17/30

(21)Application number : 11-203908
(22)Date of filing : 16.07.1999

(71)Applicant : FUJITSU LTD
(72)Inventor : KANEMASA YASUHIKO
KUBOTA KAZUMI
ISHIKAWA HIROSHI

(54) STORE AND RETRIEVAL METHOD OF XML DATA, AND XML DATA RETRIEVAL SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To make storable XML data into a data base and to make executable a complicated inquiry at a high speed.
SOLUTION: A relation data base of an XML data store means 1 includes an intermediate node table 2 which stores the intermediate node information, a link table 3 which stores the link information, a leaf node table 4 which stores the leaf nodes, an attribute table 5 which stores the attribute information, a path ID table 6 where the path IDs are made to correspond to the character strings and a label ID table 7 where the label Ids are made to correspond to the character strings. The XML data which are expressed in a tree structure are divided into nodes, and these nodes are made to correspond to the link information and stored in the tables 2–7. When the XML data are retrieved, an inquiry statement is given to an inquiry processing means 9. The means 9 executes an inquiry to track a tree structure by using index 8 and outputs a requested retrieval result.



LEGAL STATUS

[Date of request for examination] 06.09.2002

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁(JP)　(12)公開特許公報（A）　(11)特許出願公開番号

特開２００１－３４６１９

(P２００１－３４６１９A)

(43)公開日　平成13年2月9日(2001.2.9)

(51)Int.Cl.⁷	識別記号	FI			テーマコード(参考)
G06F 17/30		G06F	15/419	320	5B075
			15/403	330 B	
				340 D	

審査請求　未請求　請求項の数５　ＯＬ　（全１５頁）

(21)出願番号　特願平11-203908

(22)出願日　平成11年7月16日(1999.7.16)

(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中4丁目1番1号

(72)発明者　金政　泰彦
神奈川県川崎市中原区上小田中4丁目1番1号　富士通株式会社内

(72)発明者　久保田　和己
神奈川県川崎市中原区上小田中4丁目1番1号　富士通株式会社内

(74)代理人　100100930
弁理士　長澤　俊一郎　（外1名）

最終頁に続く

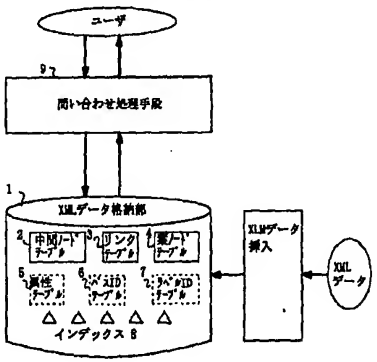
(54)【発明の名称】ＸＭＬデータの格納／検索方法およびＸＭＬデータ検索システム

(57)【要約】

【課題】　ＸＭＬデータをデータベースに格納し、複雑な問合わせを高速に実行すできるようにすること。

【解決手段】　ＸＭＬデータ格納手段１の関係データベースに、中間ノードの情報を格納する中間ノードテーブル２、リンクの情報を格納するリンクテーブル３、葉ノードの情報を格納する葉ノードテーブル４、属性情報を格納する属性テーブル５、パスＩＤと文字列とを対応付けたパスＩＤテーブル６、ラベルＩＤと文字列を対応表づけたラベルＩＤテーブル７を設け、木構造で表現されたＸＭＬデータをノード単位で分割し、上記テーブル２～７に各ノードとリンク情報を関係付けて格納する。ＸＭＬデータを検索するには、問い合わせ処理手段９に対し問い合わせ文により問い合わせを行う。問い合わせ処理手段９は、インデックス８を用いて木構造を辿る問い合わせを実行し、要求された検索結果を出力する。

本発明の基本構成図



１

【特許請求の範囲】

【請求項１】　ＸＭＬで記述されたデータを、エレメントを中間ノードとし、エレメント値と属性値を葉ノードとし、タグをリンクとする木構造で表現し、

ＸＭＬの木構造をノードとリンクに分解し、各ノードとリンク情報を関係付けて関係データベースのテーブルに格納し、

上記関係データベースに格納されたテーブルを利用して、任意の構造のＸＭＬデータを検索することを特徴とするＸＭＬデータの格納／検索方法。

【請求項２】　エレメントを中間ノードとし、エレメント値と属性値を葉ノードとし、タグをリンクとする木構造で表現されるＸＭＬで記述されたデータを検索するシステムであって、

上記システムは、ＸＭＬデータを格納する格納手段を備え、該格納手段の関係データベースに、少なくとも中間ノードの情報を格納するための中間ノードテーブルと、リンクの情報を格納するためのリンクテーブルと、葉ノードの情報を格納するための葉ノードテーブルとを設け、

上記ＸＭＬの木構造をノードとリンクに分解して、上記テーブルに各ノードとリンク情報を関係付けて格納し、上記テーブルを参照して木構造を辿る問い合わせを実行し、ＸＭＬデータを検索することを特徴とするＸＭＬデータ検索システム。

【請求項３】　関係データベースに、パスの文字列とパス用のＩＤの対応表であるパスＩＤテーブルと、ラベルの文字列とラベル用ＩＤの対応表であるラベルＩＤテーブルとを設けたことを特徴とする請求項２のＸＭＬデータ検索システム。

【請求項４】　リンクテーブルの中に各子エレメントがそのエレメント内で出現した順序の情報を付加し、葉ノードテーブルの中に各エレメント値がそのエレメント内で出現した順序の情報を付加し、上記情報により元のＸＭＬ文書の復元を可能としたことを特徴とする請求項２または請求項３のＸＭＬデータ検索システム。

【請求項５】　中間ノードテーブルに、ノードＩＤによる検索を高速に行なうためのインデックスと、テーブルの文書ＩＤによる検索を高速に行なうためのインデックスと、パスＩＤによる検索を高速に行なうためのインデックスを用意し、

リンクテーブルに、親ノードから子ノードを高速に検索するためのインデックスと、子ノードから親ノードを高速に検索するためのインデックスを用意し、

葉ノードテーブルに、ノードＩＤからそのノードの値を得るためのインデックスと、ある値を持つノードを検索するためのインデックスを用意し、

パスＩＤテーブルに、パスの文字列に対応するパスＩＤを検索するためのインデックスを用意し、

ラベルＩＤテーブルに、ラベルの文字列に対応するラベルＩＤを検索するためのインデックスを用意し、

上記インデックスを用いて木構造を辿る問い合わせを実行することを特徴とする請求項２，３または請求項４のＸＭＬデータ検索システム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、ＸＭＬで記述された大量のデータを関係データベースに格納し、検索するＸＭＬデータの格納／検索方法および検索システムに関し、特に、ＸＭＬ文書の構造に依存せずにあらゆるＸＭＬデータを格納できるようにし、また格納されたＸＭＬデータに対するＸＭＬの木構造を辿る問い合わせを高速に実行できるようにしたＸＭＬデータの格納／検索方法および検索システムに関するものである。

【０００２】

【従来の技術】現在、ＸＭＬデータを格納するのに用いられている手法は、大まかに次の２つのタイプに分類することができる。

①ファイル格納：ＸＭＬ文書をファイル形式のまま格納する手法。この手法は、オリジナルのＸＭＬファイルの全体あるいは一部をそのまま利用することを目的としており、そのため、ＸＭＬ文書をファイル形式のまま格納する。しかし、それだけでは、ファイルの数が増えたときに目的とするファイルを見つけ出すことが困難になるので、目的とするファイルを検索する為のインデックスも用意しておく必要がある。

【０００３】②テーブル格納：ＸＭＬを関係データベースのテーブルにマッピングして格納する手法。この手法ではＸＭＬ文書を構造化データと見なし、データベースに格納することによって高速な検索を行なうことを目的としている。そのため、この手法では、各エレメントを関係データベースのテーブルの各カラムにマッピングして格納する。ＸＭＬデータをテーブルにマッピングする為には、ＸＭＬの各エレメントをテーブルの各カラムにどのようにマッピングするかというマッピング規則が必要である。このマッピング規則はユーザが事前に指定する必要がある。

【０００４】

【発明が解決しようとする課題】ＸＭＬデータを格納する際に一番問題となるのは、そのデータ構造が一意に定まっていないという事である。特に、ＤＴＤ（文書型宣言）のないＸＭＬデータでは，どこにどのようなタグが出現するか分からず、データ構造は全く分からない。ＤＴＤのあるＸＭＬデータでさえも，ＤＴＤの中でタグの繰り返しやタグの選択、タグの再帰的な宣言が許されているので、データ構造が一意に定まらない。なお、このようなデータを半構造データと呼ぶ。このようなデータ構造の定まっていないＸＭＬデータを格納しようとすると、格納スキーマの設計が問題となる。例えば，図８に示される〔ＤＴＤ〕を持つ、サンプルＸＭＬデータ〔Ｘ

ＭＬデータ）をテーブル格納でデータベースに格納した場合を考える。なお、このサンプルＸＭＬデータは、２冊の本の情報を含む書籍目録のデータである。

【０００５】図９は上記ＸＭＬデータをテーブルに格納した様子を示す図である。図９のテーブルでは、１タプルが本１冊分の情報に相当していて、列にはＸＭＬデータ中で出現する可能性のある全てのタグがとられている。これを見ると、一見サンプルデータが問題なく格納されているかのように見える。しかし、サンプルデータのＤＴＤに書かれた定義には著者数の制限が無いのに、図９のテーブルでは著者を格納するスペースは最大２人分しか用意されていない。もしＸＭＬデータの中に著者がそれ以上存在したら、そのデータは格納できないか、格納しても情報が一部欠損することになる。このように、テーブル格納では、ＸＭＬのＤＴＤで記述される繰り返しタグを格納することができない。これは、テーブル格納ではあらかじめ格納する要素を列として指定しておく必要があるので、最大数が未定の繰り返し要素を表現できないからである。また、同じ理由で再帰的に定義されているタグも格納できない。さらに、そもそもＸＭＬデータにＤＴＤが存在しなくて、どのようなタグが出現するか分かっていないときには、テーブルの構造を決められず、全く対応できない。

【０００６】一方、ファイル格納は、ＸＭＬデータをファイル形式のまま格納するので、ＤＴＤの無いＸＭＬデータであろうと半構造のＸＭＬデータであろうと、格納できないＸＭＬデータは存在しない。しかし、それだけでは大量に格納されたデータの中から自分の求める情報だけを検索することができないので、検索用のインデックスが必要となる。インデックスの構成は目的に応じて色々と考えられ、簡単なものではタグ名と文字列の組をキーにして、そのタグに囲まれてその文字列が出現しているようなＸＭＬ文書を検索してくるというものがある。しかし、そのような簡単なインデックスでは、タグの階層構造を考慮した検索は行なえない。タグの階層構造の情報を持つようにインデックスを工夫することも考えられるが、それでもなお次のことが問題として残る。

【０００７】① インデックスがＸＭＬの木構造の全ての情報を持っていないので、ＸＭＬデータの全情報を使った検索ができない。

② インデックスが木構造を辿ることに最適化されていないので、そのような検索を行なった場合は検索速度が遅い。

以上のように、データ構造が一意に定まっていないＸＭＬデータにおいては、いかにしてＤＴＤ無しのＸＭＬデータや半構造のＸＭＬデータを格納するか、また、格納されたＸＭＬデータに対していかにして木構造を辿るような複雑な問い合わせを高速に実行できるようにするかといった問題がある。本発明は上記した事情に鑑みなされたものであって、本発明の目的は、データ構造が一意に定まっていないＸＭＬデータをデータベースに格納し、複雑な問合わせを高速に実行することができるＸＭＬデータの格納／検索方法およびＸＭＬデータ検索システムを提供することである。

【０００８】

【課題を解決するための手段】図１は本発明の基本構成を示す図である。同図に示すように、本発明のシステムは、エレメントを中間ノードとし、エレメント値と属性値を葉ノードとし、タグをリンクとする木構造で表現されるＸＭＬで記述されたデータを検索するシステムにおいて、ＸＭＬデータを格納する格納手段１を設け、該格納手段１の関係データベースに、少なくとも中間ノードの情報を格納するための中間ノードテーブル２と、リンクの情報を格納するためのリンクテーブル３と、葉ノードの情報を格納するための葉ノードテーブル４とを設ける。そして、上記ＸＭＬの木構造で表現されたＸＭＬデータをノード単位で分割し、上記テーブル２～４に各ノードとリンク情報を関係付けて格納する。ＸＭＬでは、木構造を形成する中間ノードと、エレメントの値を持っている葉ノードとでは、格納するために最適な格納構造が異なるので、上記のようにそれぞれ最適化された別々の専用テーブルに格納するのが望ましい。このように、値を持つためのノードである葉ノードと木構造の情報を持ったためのノードである中間ノードを別々のテーブルに格納することにより、値を格納するための格納スペースを節約することが可能となる。各ノード間の接続情報を保持する為のリンクも、リンクテーブル３に格納して持っておく必要がある。また、属性情報を格納するための属性テーブル５を別途設けてもよい。さらに、中間ノードテーブル２に各ノードのルートからのフルパス情報をＩＤで記述し、パス用のＩＤと文字列の対応表をパスＩＤテーブル６として別に持つことにより、格納スペースの節約と、検索の高速化を図ることができる。同様に、リンクテーブル３のタグ名と属性ノードテーブルの属性名をＩＤで記述し、これらラベルのＩＤと文字列の対応表をラベルＩＤテーブル７として別に持つことによって、格納スペースの節約と文字列検索の高速化を図ることができる。また、リンクテーブル３の中に各子エレメントがそのエレメント内で出現した順序の情報を付加し、葉ノードテーブルの中に各エレメント値がそのエレメント内で出現した順序の情報を付加することにより、元のＸＭＬ文書の復元が可能となる。

【０００９】本発明では、ＸＭＬの木構造をそのまま格納手段１に格納するので、ＤＴＤ無しのＸＭＬデータや半構造のＸＭＬデータも格納できる。また、ＸＭＬの木構造を全てデータベース上に格納しているので、木構造の全ての情報を検索に利用することができる。しかしこれだけでは問い合わせが行なわれたときに、ノード単位に分割して格納されているＸＭＬデータの木構造を再結合するのに時間がかかり、問い合わせの実行時間が遅くなる。そこで本発明では、上記のテーブル２～７に、ＸＭＬデータへの問い合わせパターンを考慮してインデックス８を張る。これにより、ＸＭＬの木構造を辿るような複雑な問い合わせの実行を高速に行なうことを可能となる。上記ＸＭＬデータを検索するには、例えばＸＭＬデータ検索言語により、問い合わせを行う。これにより問い合わせ処理手段９は、問い合わせ文の構文チェックを行い問い合わせのための構文木を生成し、最適な実行プランを生成する。この実行プランは、木構造検索用の関数セットで記述される。この実行プランにより、上記インデックス８を用いて木構造を辿る問い合わせを実行し、要求された検索結果を出力する。

【００１０】本発明においては、次のように構成することもできる。

（１）テーブルに関係データベースの制約の機能を適用することによって、ＸＭＬの構文規則をチェックする。

（２）リンクテーブルの中に、各エレメントの同ラベルを持つ兄弟エレメント中での出現順序の情報を付加し、各ラベルの出現順序を指定した問い合わせの実行を可能とする。

（３）リンクテーブルにリンクの両節点の情報だけでなくタグ名の情報も持つことによって、タグ名を指定してリンクを辿る問い合わせを高速に実行する。

（４）属性テーブルの中の属性ノードの接続先をリンクではなくて中間ノードにすることによって、属性を条件にして木構造を辿る問い合わせを実行する際のテーブル検索回数を削減し、問い合わせの高速実行を可能とする。

（５）中間ノードテーブルのパスＩＤによる検索を高速に行なうためのインデックスをＢ＋-treeで構築する場合において、キー値をパスＩＤとノードＩＤの組とすることによってキー値の重複を無くす。

（６）中間ノードテーブルの文書ＩＤによる検索を高速に行なうためのインデックスをＢ＋-treeで構築する場合において、キー値を文書ＩＤとノードＩＤの組とすることによってキー値の重複を無くす。

【００１１】

【発明の実施の形態】以下、本発明の実施の形態について説明する。

（１）システム構成

図２は本発明の実施例のシステムの構成を示す図である。同図に示すように、本実施例のシステムは大きくわけて、ＸＭＬデータ格納部１１、ＸＭＬデータ格納部１１にＸＭＬデータを挿入するためのＸＭＬデータ挿入モジュール１２、格納されたＸＭＬデータへの問い合わせを処理する問い合わせ処理エンジン部１３から構成される。ＸＭＬデータは、ＸＭＬデータ挿入モジュール１２によって、ＸＭＬデータ格納部１１に挿入される。ＸＭＬデータ挿入モジュール１２は、ＸＭＬパーザ１２ａとローダー１２ｂから成り、ＸＭＬパーザ１２ａは入力されたＸＭＬデータを構文解析し、ＸＭＬデータの木構造を、ＸＭＬデータ格納部１１に格納できるようにノード単位に分解する。また、ローダー１２ｂは、そのノード単位に分解された木構造をＸＭＬデータ格納部１１のテーブルに挿入する。

【００１２】図３に上記ＸＭＬデータの格納処理を示すフローチャートを示す。本実施例においてＸＭＬデータの格納処理は次のように行われる。まず、ステップＳ１において、ＸＭＬファイルを読み込む。ステップＳ２において、ＸＭＬパーザにより、入力ファイルの構文解析を行う。解析が成功した場合には、ステップＳ３に行き、ＸＭＬパーザが解析結果として、ＸＭＬの木構造のノード情報とリンク情報を中間形式としてファイル出力する。また、解析が成功しない場合には、構文解析失敗としてエラー出力し処理を終了する。ステップＳ４において、生成された中間形式ファイルを読み込み、ステップＳ５において、読み込んだＸＭＬデータをローダによって関係データベースの各テーブルに挿入し、処理を終了する。また、上記挿入が成功しない場合には、データ挿入失敗としてエラー出力をして処理を終了する。

【００１３】格納されたＸＭＬデータに対する問い合わせは、ＸＭＬデータ問い合わせ言語で行なわれ、その問い合わせは問い合わせ処理エンジン１３で処理される。問い合わせ処理エンジン１３は、問い合わせ言語のパーザ１３ａ、問い合わせ最適化エンジン１３ｂ、木構造検索用ＡＰＩ（アプリケーション・プログラミング・インタフェース）１３ｃから成る。問い合わせ言語のパーザ１３ａは、入力された問い合わせ文の構文チェックを行い問い合わせのための構文木を生成する。問い合わせ最適化エンジン１３ｂは、上記構文木を基に、最適な実行プランを生成する。この実行プランは、木構造検索用ＡＰＩ１３ｃの関数セットで記述される。木構造検索用ＡＰＩ１３ｃは、ＸＭＬデータ格納部１１とのインタフェースで、ＸＭＬの木構造上での基本的な検索を行なう関数のセットである。

【００１４】次に、上記システムにおける各部の構成についてさらに詳細に説明する。

（１）テーブル構成

まず、上記ＸＭＬデータ格納部１１に格納されるテーブルの構成について説明する。ＸＭＬデータを木構造で表現する方法はいくつかあるが、本実施例では図４に示す木構造表現を想定している。図４は、前記図８に示したＸＭＬデータを木構造で表現したものである。この木構造表現において、丸い中間ノードはエレメントを表しており、ノードの親子関係がエレメントの包含関係を表している。

【００１５】また、ノードの丸の中の数字はノードＩＤを表している。ノードとノードを結ぶリンク（枝）はタグを表しており、リンクの横に書かれている文字列はタグ名を表している。三角の葉ノードはエレメントの値を

表し、四角い楽ノードはタグに付けられた属性(Attribute)を表している。値を持つのはこの２つの葉ノードだけである。ノードを分割してデータベースに格納するときに、ノードの情報だけをデータベースのテーブルに格納したのでは、木構造のノード間の繋がり，つまりリンクの情報が欠落してしまう。そこで、リンクの情報はリンクの情報としてそれを格納する専用のテーブルを用意する。またノードも、中間ノードと、エレメント値の葉ノード、属性の葉ノードとは最適な格納構造が異なるので、別々のテーブルに格納する必要がある。

【００１６】本実施例で使用するテーブルは、全部で次の６つである。

①中間ノードテーブル

これは中間ノードの情報を格納するテーブルである。ノードＩＤ(id)の他に、そのノードが含まれている文書の文書ＩＤ(docid)、そのノードまでのルートからのフルパスのＩＤ(pathid)をカラムとして持っている。

②リンクテーブル

これはノード間のリンクを格納するテーブルである。ノードＩＤ(id)、リンクのラベル（タグ名）のＩＤ(labelid)、子ノードのノードＩＤ(child)、その子ノードの全兄弟ノード中での出現順序(ord:total order)、その子ノードの同ラベルを持つ兄弟ノード中での出現順序(pord:partial order)をカラムとして持っている。上記のように、リンクテーブル中にラベル（タグ名）のＩＤ(labelid)を付加することによりタグ名を指定してリンクを辿る問い合わせを高速に実行することが可能となる。

【００１７】③葉ノードテーブル

これはエレメント値の葉ノードを格納するテーブルである。そのエレメントにあたる中間ノードのノードＩＤ(id)の他に、エレメントの値(value)と、そのエレメント中でその値が出現した順序(order)をカラムとして持っている。このように、値を持つための葉ノードテーブルを、前記中間ノードテーブルとは別に設けることにより、値を格納するスペースを節約することができる。

【００１８】④属性ノードテーブル

これはタグにつけられた属性（例えば図８における<book year="1995">におけるyear）を格納するテーブルである。そのタグが含まれるエレメントにあたる中間ノードのノードＩＤ(id)の他に、属性名のＩＤ(labelid)、属性値(Attvalue)をカラムとして持つ。なお、属性テーブルに関係データベースの制約機能を用いて、(id, labelid)の組がユニークという制約をかけておくことによって、「同一のタグ内では同一の属性名は出現してはならない」というＸＭＬの属性に関する構文規則をチェックすることができる。また、本実施例で想定している木構造表現では、ＸＭＬのタグが木構造のリンクに相当するので、ＸＭＬのタグに付けられる属性は本来ならばリンクに付くべきである。しかし、図４では、属性はリンクに対してではなく、その下のノードに付いている。これは、検索時のテーブル参照の回数を少なくするためである。すなわち、属性を条件として木構造を辿る問い合わせを実行する際のテーブル検索回数を削減し、問い合わせの高速化を図ることが可能となる。

【００１９】⑤パスＩＤテーブル

これはパスＩＤとパスの文字列の対応表である。パスの文字列を中間ノードテーブルに直接書き込まないでこのように別に持っているのは、スペースの節約の為もあるが、パス名の文字列マッチングを含む検索が行なわれたときに、検索対象が少なくてすみ、検索が高速化できるからでもある。

⑥ラベルＩＤテーブル

これはラベルＩＤとラベルの文字列の対応表である。このように、リンクテーブルのタグ名と、属性ノードテーブルの属性名をＩＤで記述し、このラベルのＩＤと文字列の対応表をラベルＩＤテーブルとして別に持つことにより、パスＩＤテーブルと同様、格納スペースの節約と、検索の高速化を図ることができる。

【００２０】また、上記のように、リンクテーブル中に、子ノードの全兄弟ノード中での出現順序(ord:total order)の情報を付加し、また、葉ノードテーブル中に，各エレメント値がそのエレメント内で出現した順序(order)の情報を付加することに、ＸＭＬデータ格納部１１に格納されるノード単位に分解されたＸＭＬデータから、元のＸＭＬ文書を復元することが可能となる。例えば、「今日は〈天気〉晴れ〈/天気〉だった。〇〇は〈場所〉デパート〈/場所〉へでかけた。」のようにタグで区切られた文章を復元することも可能になる。また、リンクテーブル中に、各エレメントの同ラベルを持つ兄弟ノード中での出現順序(pord:partial order)の情報を付加することにより、各ラベルの出現順序を指定した問い合わせを高速に実行することが可能となる。

【００２１】一例として、図８のサンプルＸＭＬデータ（図４の木構造表現）を上記のテーブル群で格納した様子を図５、図６に示す。図５は中間ノードテーブル、リンクテーブルの例を示す図である。中間ノードテーブルにおいて、例えば，第１行目のid（＝５）は図４において" 5"と記されたノードを示し、そのノードが含まれている文書の文書ＩＤ(docid)は１である。また、そのノードまでのルートからのフルパスのＩＤ(pathid)は１であり、このＩＤに対応したpathは、"bib. book. publisher. name"である。また，リンクテーブルにおいて、例えば１行目のid（＝４）は図４において、" 4"と記されたノードを示し、そのlabelidは５であり、このlabelid に対応するlabel は"name"である。また、その出現順序を示すord, pord はそれぞれ" 0"," 0" であり、子ノードは、図４で "5"と記されたノードである。

【００２２】図６は葉ノードテーブル，属性ノードテーブル、パスＩＤテーブル、ラベルＩＤテーブルの例を示す図である。葉ノードテーブルにおいて、例えば第１行目のid（＝５）は図４において、"５"と記されたノードを示し、そのorder は" 0"、またその葉ノードの値(value)は"Addison-Wesley"である。属性ノードテーブルにおいて、例えば第１行目のid（＝３）は図４において、" 3"と記されたノードを示し、そのlabelid は３("year"に対応)，その属性値（attvalue）は"１９９５"である。また，パスＩＤテーブル、ラベルＩＤテーブルにはそれぞれ、上記各テーブル中のpathid、labelid に対応したパスの文字列、ラベルの文字列が格納され、例えば、pathid=" 1"に対応した文字列は前記したように" bib. book. publisher. name"であり、また、例えばlabelid =" 1"に対応した文字列は" bib"である。

【００２３】（２）インデックスの構成

本実施例においては、本来連結されていたはずの木構造のノードが、前記したように１つ１つに分割されて関係データベースのテーブルに格納されている。このために、木構造を辿る問い合わせが行なわれた場合、問い合わせで辿る部分のリンクを連結し直すためにジョイン操作が行なわれる。このジョイン操作の速度は全体の検索速度に大きく影響するので、ジョイン操作を高速に行なえるようにインデックスを効果的に張っておく必要がある。また、問い合わせが行なわれる場合，検索条件として指定されるのは、エレメントの値，属性，パス，出現順序などである，それらの検索も高速に行なう必要があるので、そこにもインデックスを用意しておく必要がある。

【００２４】図７に、上記図５，図６に示したテーブルに張ったインデックスの一覧を示す。このインデックスは B+ -tree で張ってあり、キーが複数の属性の組からなるインデックスは、その組の先頭からの部分的な属性の組で検索に用いることもできる。なお中間ノードテーブルに張ってあるインデックスでキーが(pathid, id)のものは、あるパスに該当する全てのノードを検索してくるときに使用するものである。このインデックスのキーは、一見pathid単独で構わないように思われるかもしれない。しかしキーをpathidだけにすると、同じキー値を持つエントリが多量に発生して、B + -tree インデックスが機能しなくなる。上記のようにキー値をパスＩＤ(pathid)とノードのＩＤ(id)の組とすることにより、キー値の重複を無くすことができ、B + -tree の検索を高速に行うことができる。また、中間ノードテーブルに張ってあるインデックスでキーが(docid, id)も同様であり、文書ＩＤ(docid)とノードのＩＤ(id)の組とすることにより、キー値の重複を無くすことができ、B + -tree の検索を高速に行うことができる。

【００２５】（３）問い合わせの実行

前記したように、格納されたＸＭＬデータに対する問い合わせは、例えばＸＭＬデータの問い合わせ言語で行なわれる，ＸＭＬデータのための検索言語の一つとして検索言語ＸＱＬがある，ＸＱＬによる問い合わせ文を、例により簡単に説明する。

【００２６】

```
SELECT result:<$book. title>
FROM  book: bib. book
WHERE $book. author. lastname="Darwen";
```

この問い合わせの意味は「bib. book. author. lastnameがDarwenであるようなbib. bookについて，bib. book. titleを検索結果として得たい」という意味である。

【００２７】上記に示すように、問い合わせ文は大きく，SELECT，FROM，WHERE の３つの部分に別れている。SELECTの部分では検索結果として得たいエレメントのプロジェクションを指定する。FROMの部分では検索の対象となるエレメントを指定している。WHERE の部分では検索の条件のセレクションを指定する。上記のような問い合わせは前記したように、問い合わせ処理エンジン１３で処理される。問い合わせ処理エンジン１３では、上記のような問い合わせ文の構文チェックを行い問い合わせのための構文木を生成する。そして，該構文木を基に，最適な実行プランを生成する。この実行プランは、木構造検索用の関数セットで記述される。

【００２８】次に、上記ＸＭＬデータに対する問い合わせ処理が，どのように行なわれるかを説明する。ここでは、図８のサンプルＸＭＬデータを、ＸＭＬデータ格納部１１に格納し、前述した図５，図６に示したテーブルに挿入した場合を例として、上記のように「著者がDarwenである本のタイトルを求めよ」という問い合わせを行なった場合について説明する。この場合のテーブル検索は、次のように行われる。なお、下記１．～１０．の処理は、上記木構造検索用の関数により実行される。

【００２９】１．葉ノードテーブルを検索して、値が "Darwen" であるノードのノードＩＤ（＝１６）を得る。

２．パスＩＤテーブルを検索して、パス"bib. book. author. lastname " のパスＩＤ（＝４）を得る。

３．中間ノードテーブルを上記１．で得られたノードＩＤ（＝１６）で検索して、得られたパスＩＤ（＝４）が上記２．で得られたパスＩＤ（＝４）と一致することを確認する。

４．ラベルＩＤテーブルを検索して、ラベル"lastname" のラベルＩＤ（＝８）を得る。

５．リンクテーブルを検索して、上記１．で得られたノードＩＤ（＝１６）と上記４．で得られたラベルＩＤ（＝８）から，親ノードのノードＩＤ（＝１５）を得る。

６．ラベルＩＤテーブルを検索して、ラベル" author " のラベルＩＤ（＝７）を得る。

７．リンクテーブルを検索して、上記５．で得られたノードＩＤ（＝１５）と上記６．で得られたラベルＩＤ（＝７）から，親ノードのノードＩＤ（＝９）を得る。

８．ラベルＩＤテーブルを検索して、ラベル"title" のラベルＩＤ（＝６）を得る。
９．リンクテーブルを検索して上記７．で得られたノードＩＤ（＝９）と上記８．で得られたラベルＩＤ（＝６）から、子ノードのノードＩＤ（＝１２）を得る。
１０．葉ノードテーブルを検索して、上記９．で得られたノードＩＤ（＝１２)から、そのノードの値("Foundation for Object/Relational Database") を得る。以上のようにして得られた検索結果は、問い合わせ処理エンジン１３を介して出力され、ユーザに提示される。

【００３０】

【発明の効果】以上説明したように、本発明においては、関係データベースに、中間ノードの情報を格納するための中間ノードテーブルと、リンクの情報を格納するためのリンクテーブルと、葉ノードの情報を格納するための葉ノードテーブル等のテーブルを設け、ＸＭＬの木構造をノードとリンクに分解して、上記テーブルに各ノードとリンク情報を関係付けて格納し、上記テーブルを参照して木構造を辿る問い合わせを実行し、ＸＭＬデータを検索するようにしたので、データ構造が一意に定まっていないＸＭＬデータに対する複雑な問い合わせを高速に実行することができる。また、ＸＭＬの木構造をそのまま格納手段に格納するので、ＤＴＤ無しのＸＭＬデータや半構造のＸＭＬデータも格納することができる。さらにＸＭＬの木構造を全てデータベース上に格納しているので、木構造の全ての情報を検索に利用することができる。

【図面の簡単な説明】

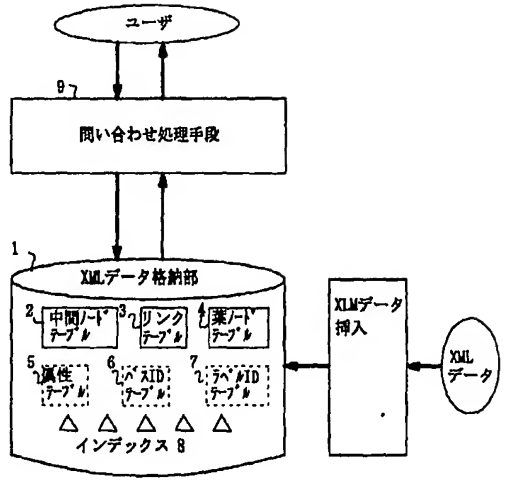
【図１】本発明の基本構成図である。
【図２】本発明の実施例のシステムの構成例を示す図である。
【図３】本発明の実施例のシステムにおける格納処理フローを示す図である。
【図４】ＸＭＬデータの木構造表現の一例を示す図である。
【図５】本発明の実施例のテーブル構成の一例を示す図（１）である。
【図６】本発明の実施例のテーブル構成の一例を示す図（２）である。
【図７】本発明の実施例のイッデックス一覧を示す図である。
【図８】ＸＭＬデータの一例を示す図である。
【図９】図８のＸＭＬデータをテーブルに格納した様子を示す図である。

【符号の説明】

１	ＸＭＬデータ格納格納手段
２	中間ノードテーブル
３	リンクテーブル
４	葉ノードテーブル
５	属性テーブル
６	パスＩＤテーブル
７	ラベルＩＤテーブル
８	インデックス
９	問い合わせ処理手段
１１	ＸＭＬデータ格納部
１２	ＸＭＬデータ挿入モジュール
１２ａ	ＸＭＬパーザ
１２ｂ	ローダ
１３	問い合わせ処理エンジン部
１３ａ	問い合わせ言語のパーザ
１３ｂ	問い合わせ最適化エンジン
１３ｃ	木構造検索用

【図１】

本発明の基本構成図



【図７】

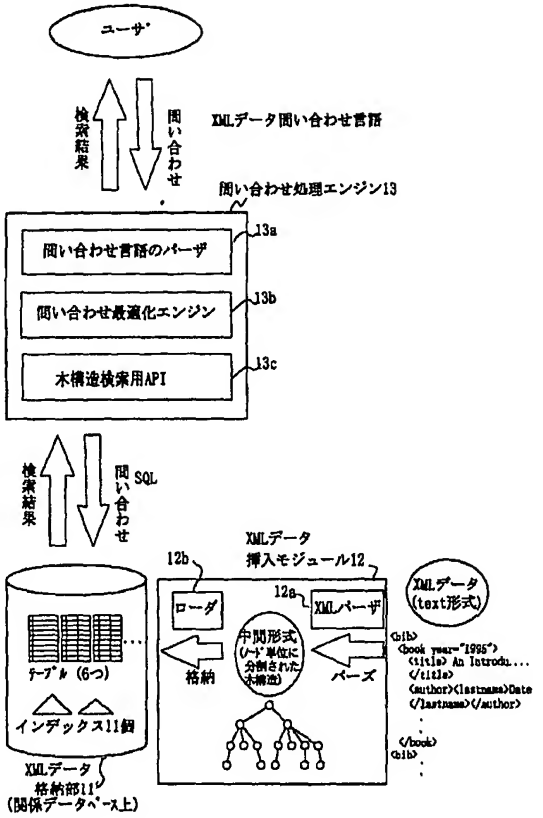
本発明の実施例のイッデックス一覧を示す図

インデックス一覧

テーブル名	キー
中間ノード	id
中間ノード	(docid, id)
中間ノード	(pathid, id)
リンク	(id, labelid, child)
リンク	(child, labelid, id)
葉ノード	id
葉ノード	value
属性ノード	id
属性ノード	attvalue
パスID	path
ラベルID	label

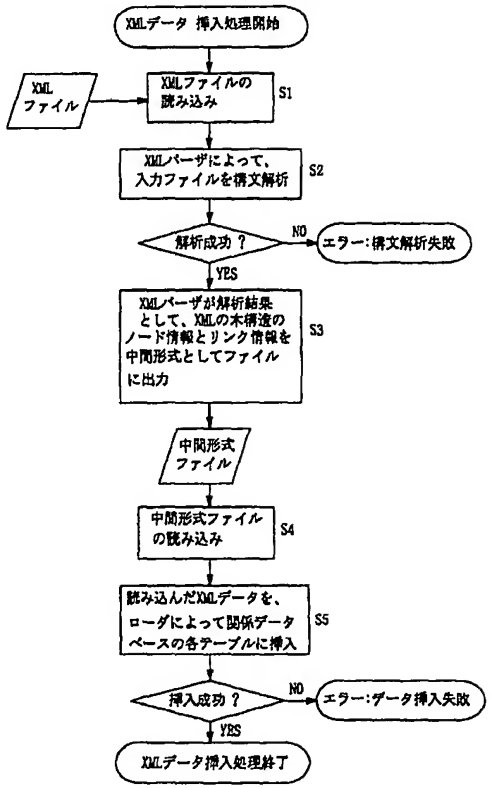
【図２】

本発明の実施例のシステムの構成例を示す図



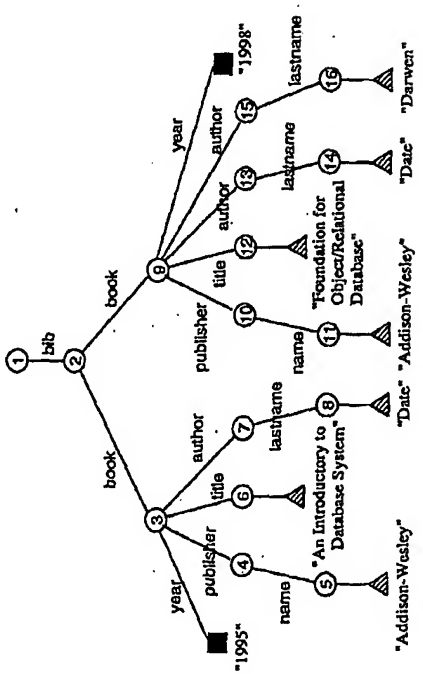
【図３】

本発明の実施例のシステムにおける格納処理フローを示す図



【図４】

XMLデータの木構造表現の一例を示す図



【図５】

本発明の実施例のテーブル構成の一例を示す図（１）

中間ノードテーブル

id	docId	pathid
5	1	1
4	2	2
6	1	3
8	1	4
7	1	5
3	1	6
11	1	1
10	1	2
12	1	3
14	1	4
13	1	5
16	1	4
15	1	5
9	1	6
2	1	7
1	1	8

リンクテーブル

id	labelid	tord	pord	child
4	5	0	0	5
7	8	0	0	8
3	4	0	0	4
3	6	1	0	6
3	7	2	0	7
10	5	0	0	11
13	8	0	0	14
15	8	0	0	16
9	4	0	0	10
9	6	1	0	12
9	7	2	0	13
9	7	3	1	15
2	2	0	0	3
2	2	1	1	9
1	1	0	0	2

【図６】

本発明の実施例のテーブル構成の一例を示す図（２）

葉ノードテーブル

id	order	value
5	0	Addison-Wesley
6	0	An Introductory to Database System
8	0	Date
11	0	Addison-Wesley
12	0	Foundation for Object/Relational Database
14	0	Date
16	0	Darwen

属性ノードテーブル

id	labelid	attvalue
3	3	1995
9	3	1998

パスIDテーブル

pathid	path
1	bib.book.publisher.name
2	bib.book.publisher
3	bib.book.title
4	bib.book.author.lastname
5	bib.book.author
6	bib.book
7	bib
8	/

ラベルIDテーブル

labelid	label
1	bib
2	book
3	year
4	publisher
5	name
6	title
7	author
8	lastname

【図８】

XMLデータの一例を示す図

[DTD]

```
<!ELEMENT book (author+, title, publisher)>
<!ATTLIST book year CDATA>
<!ELEMENT article (author+, title, year?, (shortversion | longversion))>
<!ATTLIST article type CDATA>
<!ELEMENT publisher (name, address?)>
<!ELEMENT author (firstname?, lastname)>
```

[XMLデータ]

```
<bib>
    <book year="1995">
        <title> An Introductory to Database System </title>
        <author> <lastname> Date </lastname> </author>
        <publisher> <name> Addison-Wesley </name> </publisher>
    </book>

    <book year="1998">
        <title> Foundation for Object/Relational Database </title>
        <author> <lastname> Date </lastname> </author>
        <author> <lastname> Darwen </lastname> </author>
        <publisher> <name> Addison-Wesley </name> </publisher>
    </book>
</bib>
```

【図９】

図８のXMLデータをテーブルに格納した様子を示す図

ｂｏｏｋのテーブル

ID	title	author1 firstname	author1 lastname	
1	An Introductory to Database System		Date	
2	Foundation for Object/Relational Database		Date	
：	：	：	：	

author2 firstname	author2 lastname	publisher name	publisher address	year
		Addison-Wesley		1995
	Darwen	Addison-Wesley		1998
：	：	：	：	：

フロントページの続き

(72)発明者 石川 博
神奈川県川崎市中原区上小田中４丁目１番
１号 富士通株式会社内

Ｆターム(参考) 5B075 ND36 PP23 QR00 QT06